UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Corso di laurea in INGEGNERIA INFORMATICA

Ricostruzione automatica di pose di cani da filmati 2D

Relatore:
Enoch Peserico

Candidato:
Alberto Ursino

Data laurea 22/07/2021

ANNO ACCADEMICO 2020/2021

Indice

Capitolo 1

Introduzione

1.1 Problema

In Italia si possono contare almeno 8 milioni di cani [4], per una densità pari a 26 per chilometro quadrato; un valore non del tutto trascurabile (basti pensare che la densità di popolazione è di circa 200 abitanti per chilometro quadrato). Molti di loro vengono regolarmente portati fuori a passeggio; ma alcuni padroni lasciano al proprio cane libero sfogo di imbrattare (senza poi ripulire) marciapiedi, muri delle case, beni culturali, ecc. L'esperienza di tutti i giorni di chi vive soprattutto nelle grandi città (dove poche sono le zone dedicate ai cani) può confermare l'esistenza di questi problemi: marciapiedi imbrattati, odori sgradevoli, muri delle proprie case macchiati, ecc. Per le forze dell'ordine non è facile cogliere sul fatto certi comportamenti e sanzionarli: l'atto è quasi istantaneo e quindi difficile da notare.

Soluzioni che fanno fronte a questo problema già esistono: l'utilizzo di repellenti, l'affissione di avvisi, l'installazione di zone adatte ai cani, ecc. Queste e molte altre sono soluzioni che aiutano ma che non sono altamente efficienti; per questo abbiamo pensato di crearne una con l'aiuto dell'intelligenza artificiale.

1.2 Idea di soluzione e struttura della tesi

Per rispondere a questo problema abbiamo progettato un sistema automatizzato in grado di riconoscere in tempo reale eventuali comportamenti scorretti assunti dal cane. Il nostro obiettivo è quindi sostituire il lavoro che farebbe un essere umano se dovessimo assumerlo per controllare 24/7 l'uscio di casa nostra. A questo scopo, è necessario addentrarsi nel campo del deep learning nel quale troviamo il mondo delle reti neurali artificiali. Il cervello del nostro sistema è proprio una rete di questo tipo e lo scopo del nostro progetto è implementarne una per poter sostituire appunto il lavoro dell'essere umano.

In questa tesi descriveremo il nostro progetto passo dopo passo partendo da un breve accenno sulla teoria delle reti neurali artificiali, utile per rendere più chiari i loro meccanismi di funzionamento. Nei capitoli principali presenteremo il nostro lavoro mostrando dati concreti e le nostre varie scelte progettuali. In fine (oltre alle

conclusioni) illustreremo le migliorie possibili relative a questo progetto.

Prima di iniziare col capitolo delle reti neurali, dobbiamo precisare ancora due cose. Alcune parti del progetto sono attualmente ancora in fase di sviluppo; in corrispondenza di alcuni argomenti sarà infatti scritto esplicitamente che verranno ripresi nel capitolo delle direzioni future 5.2. Per concludere, riteniamo necessario presentarvi il mio cane, Spot. Più volte (all'interno di questa tesi) verrà citato e preso in esempio. Grazie a lui abbiamo potuto ottenere risultati fondamentali utili al raggiungimento dell'obiettivo.

Capitolo 2

Reti neurali

2.1 Definizione

Le reti neurali artificiali (in inglese Artificial Neural Network) sono sistemi artificiali ispirati alla rete neurale umana [7]. I compiti di una rete neurale biologica sono fondamentalmente due: rispondere a stimoli esterni e memorizzare informazioni. Grazie alla capacità delle ANN di riuscire a modellarsi in base ad una funzione obiettivo, riescono anch'esse a svolgere tali compiti (entro certi limiti). Come si può intuire, il loro scopo è replicare il funzionamento del cervello umano con l'obiettivo di poterlo sostituire in certe realtà.

La potenza delle ANN non è ancora paragonabile a quella del cervello umano, per questo abbiamo specificato "entro certi limiti". Il campo di utilizzo quindi è sempre limitato ad una certa realtà. Vengono implementate ad esempio nel campo automobilistico per rendere possibile la guida autonoma o nel campo medico per prevedere eventuali patologie con l'analisi del DNA. Attualmente, il campo di ricerca delle ANN è in continuo sviluppo; in futuro, probabilmente, potranno raggiungere la complessità e la potenza del cervello umano, se non superarla.

2.2 Percettrone

Gli elementi funzionali principali delle rete neurali artificiali sono i nodi (che di fatto sono dei neuroni artificiali); essi sono progettati per funzionare in maniera simile ai neuroni biologici. Uno dei primi neuroni artificiali è il percettrone, inventato da Frank Rosenblatt nel 1958 [9] e ancora oggi usato come modello di riferimento teorico.

La struttura classica di un percettrone è rappresentata in fig.2.1. Le variabili $x_1, x_2, ..., x_n$ sono i suoi input e rappresentano segnali inviati da altri percettroni. La variabili $w_1, w_2, ..., w_n$ sono i pesi relativi ad ognuno degli input; questi hanno un ruolo molto importante nella fase di addestramento (che vedremo nella sezione successiva). Il nucleo del percettrone è composto da due parti: una sommatoria Σ e una funzione di attivazione $f(z)$. Rispettivamente, la prima determina un valore

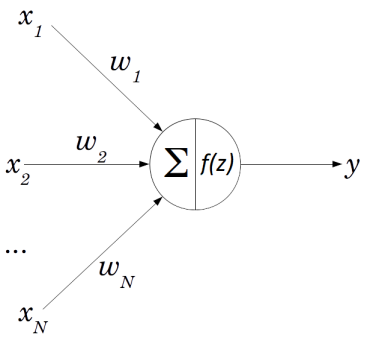


Figure 2.1: *Struttura di un percettrone*

z a partire dagli input e i loro pesi (eq.2.1).

$$\sum_{i=1}^{N} x_i * w_i = z \tag{2.1}$$

La seconda decide se z è un valore abbastanza "forte" per far attivare il percettrone; in tal caso viene generato un valore di output y che rappresenta altra informazione. La funzione di attivazione può essere ad esempio la classica funzione a gradino (eq.2.2).

$$y = f(z) = \begin{cases} 1 & \text{se } z \geq 0 \\ 0 & \text{se } z < 0 \end{cases} \tag{2.2}$$

L'output è collegato ad altri percettroni in modo tale da far continuare il ciclo e formare così la rete neurale. Esistono diverse strutture di collegamento tra i neuroni che vanno quindi a diversificare le varie tipologie di ANN.

2.3 Addestramento

Il processo fondamentale che permette di dare uno scopo alle reti neurali artificiali è l'addestramento. Nel campo del deep learning, l'obiettivo principale è addestrare una rete neurale artificiale a produrre un certo output da un dato input. Per rendere più chiaro il concetto faremo questo semplice esempio con una rete neurale umana: i giocatori di tennis più esperti riescono a capire la dinamica della palla al rimbalzo solamente guardando in che modo l'avversario la colpisce. Il ragionamento che viene fatto dalla loro rete neurale si compone banalmente di tre fasi: nella prima viene ricevuta in input la sequenza di movimenti dell'avversario prima, al momento e dopo il colpo; nella seconda viene attuata un'analisi (quasi istantanea) dell'input appena ricevuto e nella terza fase viene generato l'output che rappresenta un'idea della dinamica della palla al momento del rimbalzo. Questa idea è più o meno precisa a seconda di quanta esperienza la rete neurale del tennista ha accumulato durante la sua vita. In altre parole, il tennista è più bravo a predire il rimbalzo quanto più ha fatto esperienza durante gli allenamenti, partite, ecc. Lo stesso concetto è

applicabile alle reti neurali artificiali: esse vengono addestrate per capire e analizzare una certa realtà allo scopo di produrre risultati utili e il più precisi possibile.

L'addestramento viene effettuato fornendo alla rete esempi della realtà sotto studio sotto forma di input e i relativi output. Gli esempi che forniamo alla rete sono rappresentati da dati che possono essere di diverse tipologie (e.g. singole immagini, sequenze di immagini, suoni, valori numerici, frasi, ecc.). Il tipo corretto di dato va scelto in base alla realtà che si vuole studiare. Se volessimo allenare un'ANN allo scopo di predire la dinamica della palla da tennis a rimbalzo, dovremmo fornirle numerosi esempi del tipo: come input sequenze di immagini rappresentanti l'avversario che colpisce la palla e come output, banalmente, un valore numerico che contraddistingue il corrispettivo tipo di rimbalzo (1 = palla morbida con spin contro, 2 = palla veloce con spin a favore, ecc.). Tutti i dati utili ad addestrare la rete fanno parte del dataset (a cui è dedicata la sezione 3.1); esso è generalmente suddiviso in due parti: il training set e il validation set. Il training set contiene i dati su cui viene modellata la rete ed è di fatto la base di ogni progetto nel campo del deep learning; il suo scopo è quello di addestrare la rete sulla realtà che vogliamo studiare. Il validation set contiene dati esterni al training set ed è utile per effettuare una valutazione delle prestazioni del modello. In particolare, serve a capire come si comporterebbe la rete in campo reale, cioè con dati mai visti prima. Con il testing sul validation set si possono notare alcune criticità del modello che permettono di capire come correggere il training set iniziale e migliorare così l'addestramento.

Le ANN vengono addestrate con l'ausilio di un algoritmo di addestramento capace di modificare i pesi interni al modello (di cui parlavamo nella sezione 2.2). Un'ANN viene modellata con l'obiettivo di trovare un pattern logico tra gli input e i relativi output che abbiamo fornito. Alla fine dell'addestramento, la rete dovrà essere capace di processare un output corretto dato solo l'input. Il processo di modellamento viene ottimizzato da una funzione obiettivo, chiamata *loss function*; essa ha lo scopo di calcolare quanto si discostano le predizioni fatte dal modello con quelle teoriche, quindi tanto più piccola è, tanto meno la predizione è affetta da errore. Un esempio di questa funzione è la *Mean Squared Error* (eq.2.3).

$$L(y, y') = \frac{1}{N} * \sum_{i=0}^{N} (y - y'_i)^2 \tag{2.3}$$

Sostanzialmente, l'algoritmo di addestramento con supporto la *loss function* inizia col fornire alla rete gli input sui quali la rete effettuerà delle predizioni. Una volta prodotto l'output, viene confrontato con quello teorico (che abbiamo creato noi) e viene così calcolata la *loss function*. Sulla base del valore ottenuto, vengono ricalcolati i pesi interni al modello. Questo processo viene ripetuto per diverse iterazioni fino a che la *loss function* satura ad un certo valore: significa cioè che la rete è riuscita a modellarsi in modo tale da minimizzarla. Quando il valore saturato è 0 vuol dire che la rete è riuscita a trovare il pattern logico perfetto tra gli input e gli output. A questo punto, se il modello addestrato rispetta determinati criteri è possibile impiegarlo nell'ambito reale voluto. Se il modello trainato presenta qualche

problema (ad esempio è affetto da overfitting[1]) di solito viene riguardato il TS o modificati gli iperparametri[2].

2.4 Stacked Hourglass Network

La rete neurale artificiale che abbiamo usato nel nostro progetto è la Stacked Hourglass Network, un tipo di rete convoluzionale. Le SHN vengono largamente usate per la stima della posa di un corpo [8] ed è per questo che abbiamo deciso di farci affidamento. Il modello teorico della rete è rappresentato in fig.2.2.

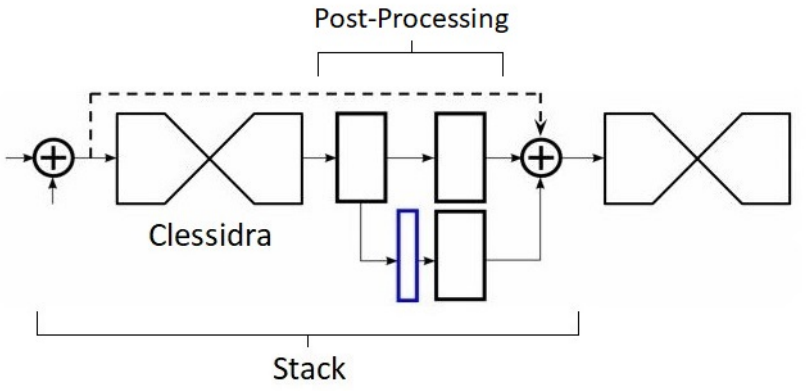


Figure 2.2: *Modello teorico della SHN (Source: [8])*

Di seguito presenteremo la struttura di una SHN da un punto di vista generale, senza entrare nel dettaglio. Come si può notare dalla fig.2.2, il pezzo principale del modello è lo stack, il quale è composto da una "clessidra" e dei blocchi adibiti al post-processing. Una *stacked* hourglass network è formata da più stack, posti uno dopo l'altro; mentre è possibile impostare un'hourglass network anche con un solo stack (che è il tipo di struttura utilizzata da noi). La clessidra (fig.2.3) è composta da blocchi di diverse dimensioni (ognuno con una propria funzione) e la loro volta formati da diversi strati di neuroni. La clessidra ha inoltre la caratteristica di essere suddivisa in due parti funzionali: la codifica e la decodifica. Entrambe hanno gli stessi blocchi posti specularmente che sono oltretutto collegati per favorire lo scambio di informazioni. Successivo alla clessidra vi è un sistema di blocchi per il post-processing che forma la parte finale dello stack.

Il processo di predizione ha inizio con la ricezione in input di un'immagine. Questa viene inviata alla parte di codifica della clessidra dove ne vengono estratte

[1]Quando un modello si adatta strettamente alle caratteristiche specifiche del training set, quindi non è capace di effettuare previsioni corrette su dati non visionati, siamo in presenza del problema di overfitting. Tale complicazione si riscontra nella maggior parte dei casi in cui il training set non è generalmente ben formato.

[2]Gli iperparamerti sono parametri regolabili che consentono di controllare il processo di training del modello. Ad esempio, con le reti neurali si decide il numero di livelli nascosti e il numero di nodi in ogni livello. Le prestazioni del modello dipendono in gran parte dagli iperparamersi.

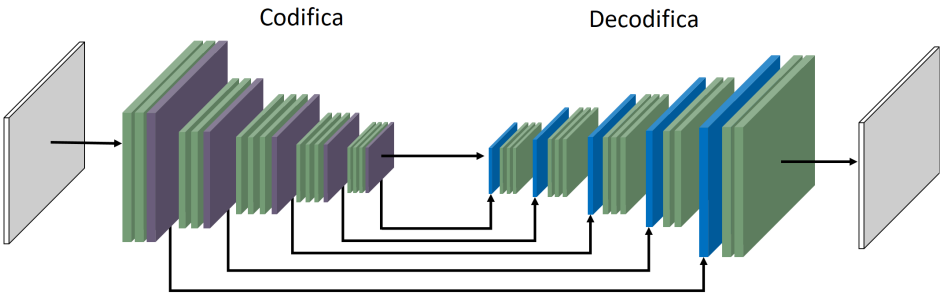


Figure 2.3: *Struttura della clessidra [Sunner Li - Simple Introduction about Hourglass-like Model]*

le feature[3]. Successivamente (nella parte di decodifica) le feature vengono analizzate e combinate per formare un'idea più chiara e dettagliata dell'immagine. La comunicazione tra i vari blocchi delle due parti funzionali ha lo scopo di migliorare la comprensione generale dell'immagine. In seguito, il risultato di tutte le analisi fatte nella clessidra passa attraverso il sistema di blocchi finale. Questo effettua una sorta di post-processing allo scopo di generare le heatmap grazie alle quali la rete è in grado di restituire in output una predizione.

Programmare una Stacked Hourglass Network è una procedura piuttosto complicata per via della sua non banale struttura; inoltre non è il target principale di questo progetto. Detto questo, abbiamo deciso di far affidamento su Deepposekit[6], una repository che mette a disposizione una SHN completamente funzionante. La rete in questione è stata inoltre assemblata allo scopo di riconoscere la posa di un animale da un video 2D, che è proprio quello che ci serve.

[3]Nel deep leanring, la feature è una proprietà individuale e misurabile di un fenomeno osservato. La scelta di caratteristiche discriminanti, ad alto contenuto informativo e indipendenti fra loro è un passo cruciale per ottenere un efficiente algoritmo di riconoscimento di pattern, classificazione e regressione.

Capitolo 3

Il nostro lavoro

3.1 Dataset

(a) Immagine RGB di Spot (b) Silhouette di Spot (c) Silhouette di Spot annotata

Figure 3.1: *Processo di creazione di un dato del training set*

Il nostro training set è formato da 298 immagini di Spot; tutte annotate manualmente (grazie ad un tool implementato da Deepposekit) secondo il modello a scheletro con joint. Come abbiamo spiegato nella sezione 2.3, per insegnare alla rete a capire una certa realtà, bisogna fornirle un input e il relativo output. Nel nostro caso, come input forniamo delle silhouette di Spot e come output le stesse silhouette però annotate. Per l'appunto, il nostro obiettivo è dare in pasto alla rete un'immagine di un cane (in particolare la sua silhouette) e farci restituire la stessa con evidenziata la posa.

Il modello a scheletro con joint è essenziale per rappresentare matematicamente la posa di un cane; viene infatti utilizzato ampiamente nei progetti di deep learning per affrontare problemi legati alle pose in generale [8], I joint sono punti particolari che vanno ad indicare le varie parti del corpo del cane. I collegamenti tra i vari joint formano così lo scheletro (da qui il nome del modello). Le parti del corpo che abbiamo ritenuto importanti per il nostro studio sono: la coda, le zampe (sia posteriori che anteriori), il collo, il naso, la mascella e le orecchie. A questo proposito, abbiamo impostato il modello in modo tale da avere 3 joint per la coda, 3 per ogni zampa, e 1 per ogni restante parte del corpo. Un esempio di un'immagine annotata con questo modello lo si ha in fig.3.1 (c) che rappresenta oltretutto un dato effettivo del nostro TS. Le posizioni di Spot che abbiamo preferito sono le stesse che la rete vedrà più spesso in campo reale; quindi lui visto posteriormente, anteriormente e mentre fa i propri bisogni.

L'utilizzo delle silhouette ha lo scopo di rendere la realtà meno complessa agli occhi della rete neurale. Le immagini RGB come in fig.3.1 (a) contengono molta più informazione e questo può confondere la rete soprattutto se il TS non è ampio, come nel nostro caso. Per una prima implementazione quindi, abbiamo ritenuto sufficiente un addestramento con silhouette. Per la loro estrazione abbiamo usato una versione del software DeepLabv3+ [3] preimpostata [DEEPLABV3-RESNET101], quindi non funzionale al 100% per la nostra realtà (di questo parleremo meglio nella sezione 4.2). Tale software estrae le silhouette di diversi soggetti (cani, gatti, persone, ecc.) e noi abbiamo fatto in modo di ottenere in output la sola silhouette del cane, settando il colore delle altre silhouette estratte di nero (lo stesso dello sfondo).

Le immagini del TS sono state prese tutte dalla stessa angolazione. La stessa su cui dovrà essere impostata (idealmente) la telecamera in campo reale (parleremo meglio dei dispositivi nella sezione ad essi relativa 3.2.1). Nella realtà, la rete dovrà fare predizioni su immagini di cani mai viste prima; il fatto quindi di standardizzare l'angolazione della telecamera, permette di catturare immagini più o meno simili a quelle del TS. Facendo ciò, la rete riuscirà a predire con maggiore accuratezza.

La scelta di utilizzare solo immagini di Spot è stata fatta dopo aver tentato di creare un TS con immagini di cani diversi. Inizialmente avevamo realizzato un training set composto da più di 700 immagini (contenente sia immagini di Spot sia immagini di altri cani prese dal web). Ci siamo accorti che la rete non riusciva a generalizzare sufficientemente la realtà e si presentava marcatamente il problema di overfitting. Secondo noi, il motivo principale che causava tale problema era l'eccessiva eterogeneità di immagini. Non solo erano presenti immagini di cani in diverse posizioni (tutte le possibili), ma era soprattutto presente la non banale varietà canina. Tutto questo rendeva la realtà di una complessità troppo elevata non permettendo alla rete di generalizzarla; facendo così che imparasse strettamente sul training set. Nella sezione 5.2.1 spiegheremo delle migliorie possibili relative al training set.

Il nostro validation set è formato da immagini prese direttamente dal TS. Viene creato grazie a Deepposekit (prima di ogni sessione di addestramento) il quale estrapola 1 immagine ogni 10 dal TS.

3.2 Implementazione

3.2.1 Dispositivi teorici

Il dispositivo più importante (su cui si basa l'intero sistema) sarà un elaboratore. Lo scopo di questo dispositivo dovrà essere quello di fornire tutti i componenti hardware e software necessari a far girare la rete neurale artificiale, l'estrattore di silhouette e la parte applicativa legata all'analisi delle predizioni. Ad esso saranno collegati tutti gli altri dispositivi in modo tale da rendere possibile lo scambio di dati e comandi.

Per permettere alla rete di effettuare predizioni in tempo reale sarà necessaria una telecamera in grado di fornire costantemente immagini RGB (i dati fondamentali su cui lavora l'intero sistema). Ci sono due aspetti importanti da tenere in considerazione riguardo la telecamera: l'angolazione e la risoluzione su cui impostarla.

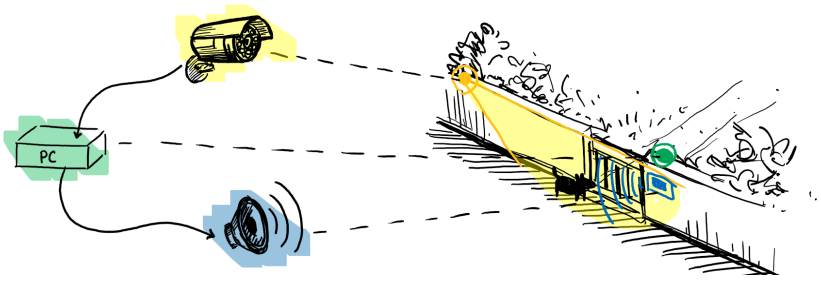


Figure 3.2: *Esempio approssimativo del posizionamento dei dispositivi in un ambiente reale (Schema illustrativo creato da Claudia Martinazzo, 2021)*

La telecamera dovrà essere innanzitutto installata in modo tale da riprendere la zona d'interesse (e.g. l'uscio di casa) e l'angolazione regolata adeguatamente. In particolare, dovrà essere più possibile uguale a quella usata per le immagini del TS (come abbiamo già detto nella sezione 3.1). Un esempio di angolazione ideale lo si ha nell'immagine (a) in fig.3.1; come si può notare, la telecamera ha ripreso il cane leggermente dall'alto e ad una distanza relativamente vicina. Se il cane dovesse trovarsi troppo distante nella scena, l'estrazione della silhouette risulterebbe imprecisa. I formati ideali su cui poter impostare la telecamera sono il 2:1 e il 16:9. La risoluzione non deve essere troppo bassa per evitare estrazioni imprecise. Una risoluzione accettabile (e testata) è la 1920x1080.

Infine, per rendere utile il nostro sistema sarà necessario un dispositivo in grado di attuare misure concrete laddove è necessario. Per una scelta progettuale, non ci occupiamo di implementare tale dispositivo ma forniamo solo il sistema che formula la decisione (su cui si dovrebbe basare teoricamente il dispositivo in questione). Un esempio vago potrebbe essere un emettitore di ultrasuoni percepibili solo dal cane (vedi elemento evidenziato in blu in fig.3.2).

3.2.2 Componenti software

Subito dopo aver installato e avviato i vari dispositivi vi è la fase di avviamento del sistema. Tutti i componenti software vengono inizializzati (e.g. viene caricata la rete per le predizioni) e viene aperto il canale di comunicazione tra la telecamera e l'elaboratore. Dopo un breve periodo di tempo, il sistema è online e la telecamera comincerà a generare costantemente frame RGB. Tutti i componenti software (che adesso andremo a descrivere in dettaglio) e i collegamenti logici tra di essi sono rappresentati in fig.3.3. Come si può notare, tutte le parti applicative si trovano all'interno dell'elaboratore.

La parte applicativa *frame extractor* gestisce il canale di comunicazione tra l'elaboratore e la telecamera. In particolare, essa ha lo scopo di estrapolare i frame RGB dallo streaming video generato dalla telecamera e inviarli alla parte applicativa successiva. Dopo che l'elaboratore ha "consumato" un frame RGB (cioè ha generato l'eventuale silhouette del cane, ha eseguito le predizioni e attuato l'analisi dei joint) sarà pronto per analizzare un nuovo frame. A seconda della potenza di calcolo dei componenti interni all'elaboratore quindi, la frequenza di predizione varierà

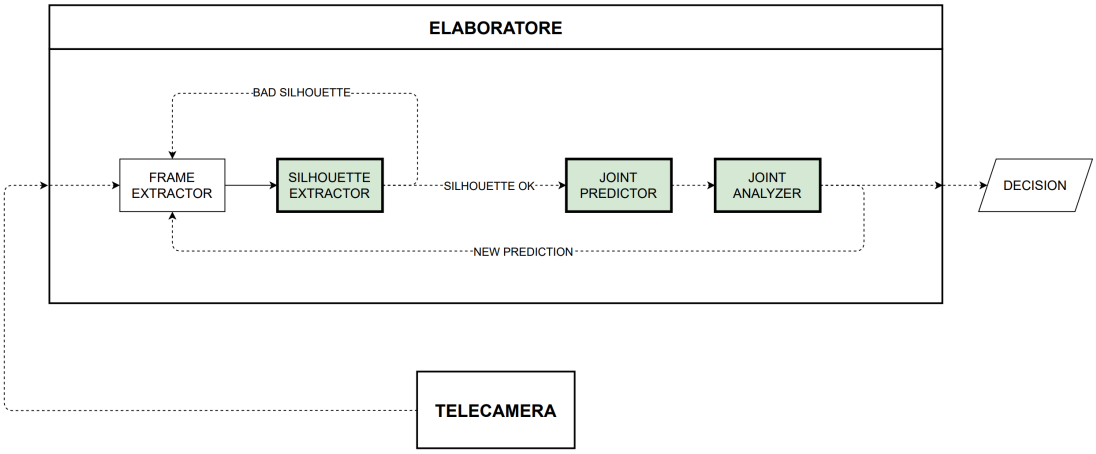


Figure 3.3: *Grafico logico descrivente i vari componenti software del sistema. In verde le parti applicative più complesse*

di conseguenza. Se l'elaboratore impiegasse 0.1 secondi per analizzare un frame significa che la frequenza video ideale su cui settare la telecamera sarebbe 10Hz. Ad ogni modo, il *frame extractor* che abbiamo ideato noi è capace di estrapolare un frame RGB appena possibile, permettendo di settare la telecamera ad una frequenza qualsiasi.

Il frame video appena catturato viene mandato in input al *silhouette extractor* il quale ha lo scopo di estrapolare la silhouette di un cane. Se estratta con successo, il programma farà un resize dell'immagine alla risoluzione 512x256 in modo tale da renderla leggibile dalla rete al passo successivo. Ricapitolando, l'output di questo applicativo è un'immagine di risoluzione 512x256 raffigurante la silhouette di un cane (vedi fig.3.1 (b)). Tale parte applicativa non sempre potrà adempiere al suo obiettivo, basti pensare alla situazione in cui non è presente alcun cane nella scena; in questo caso sarà possibile ricominciare il ciclo quindi inviare un segnale al *frame extractor* il quale estrapolerà un nuovo frame RGB.

Nella fase successiva, l'applicativo *joint predictor* riceve in input l'immagine precedentemente generata dal *silhouette extractor*. Tale immagine viene analizzata dalla rete neurale artificiale la quale effettuerà le previsioni. Per com'è progettato, questo applicativo restituisce in output un'immagine 512x256 raffigurante la silhouette di un cane integrata con lo scheletro predetto.

Nell'ultima fase viene eseguita l'analisi delle predizioni dall'applicativo *joint analyzer* il quale valuta la posizione del cane: "va segnalata o il cane sta semplicemente passeggiando?". Tale parte applicativa è attualmente ancora in fase di sviluppo. Nella sezione 5.2.3 descriveremo delle idee di progettazione ad essa relative.

3.2.3 Impostazioni

Come abbiamo già specificato nella sezione 2.4, il modello di rete utilizzato è la stacked hourglass network. Per i suoi settaggi strutturali ci siamo ispirati a BADJA [2], un progetto simile al nostro. La nostra rete è quindi formata da 1 stack, 7 layers

intermedi e una matrice filtro di 256x256.

Per le sessioni di training (con il TS definito nella sezione 3.1) abbiamo impostato un batch size di 4 e per il validation set un batch size di 10. Abbiamo trainato il modello per 1000 epoche sapendo però che la sessione di addestramento sarebbe terminata prima grazie alla funzione *early stopping* (implementata da Deepposekit). Questa funzione ha lo scopo di terminare il training ad un'epoca precedente a quella impostata se la rete smette di imparare; quindi analizza l'andamento della *loss function*.

L'ambiente di sviluppo del progetto è caratterizzato dall'utilizzo di Python (ver. 3.6) con sistema operativo Windows 10. Le piattaforme principali che abbiamo utilizzato per implementare la rete neurale artificiale sono Tensorflow-GPU (ver. 1.14.0) e Keras (ver. 2.2.5).

Come hardware di supporto abbiamo avuto a disposizione una scheda video GTX 1660 Ti con 6gb di memoria, un processore Ryzen 2600X (4.25Ghz) e 16gb di RAM. Per fornire la compatibilità necessaria tra Tensorflow-GPU e la GPU stessa abbiamo installato i package cuDNN (ver. 7.4) e CUDA (ver. 10). Ad assicurare la completa compatibilità tra tutti i package, piattaforme, ecc. abbiamo utilizzato il software Anaconda [1].

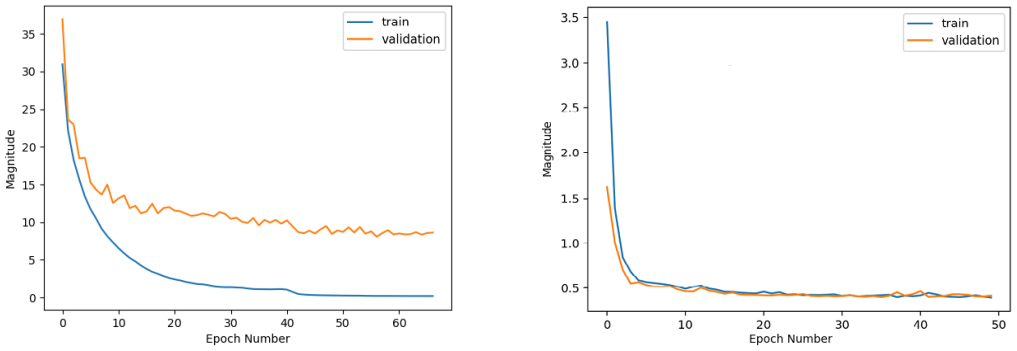
Capitolo 4

Risultati e prestazioni

Di seguito presenteremo i dati che abbiamo estrapolato dalle varie parti applicative allo scopo di studiarne le prestazioni.

4.1 Fase di addestramento

I dati utili relativi alla fase di training riguardano sicuramente la *loss function* (spiegata nella sezione 2.3), la *val loss function* (essenzialmente, la loss function calcolata sul validation set) e la *confidence function* (spiegata in seguito).



(a) Andamento sperimentale della *loss function* e della *val loss function*

(b) Andamento teorico ideale della *loss function* e della *val loss function* [Source]

Figure 4.1: *Grafici delle curve relative alla più recente sessione di training*

Osservando il grafico relativo alla nostra *loss function* 4.1 (a) (che abbiamo ottenuto durante l'ultima sessione di training) si può notare che parte da un valore relativamente alto, evidenziando il fatto che la rete non ha idea di come formulare un output sensato avendo tra le mani la sola silhouette del cane. Con l'avanzare delle epoche (che rappresentano le iterazioni di addestramento) il valore della *loss function* tende a saturare a 0, rispettando le previsioni teoriche. Guardando solo a questa funzione si potrebbe pensare che la rete sia ben addestrata; per effettuare una pressoché completa valutazione delle prestazioni però, va analizzata anche la *val loss function*. Un andamento ideale di tale funzione lo si può osservare in figura

4.1 (b): dovrebbe quindi anch'essa saturare a 0, seguendo l'andamento della *loss function*. Questo comportamento ideale è nella maggior parte dei casi segno di una rete ben addestrata capace di fare previsioni precise su dati mai visti. Ritornando al nostro grafico, possiamo notare come la *val loss function* satura invece ad un valore maggiore di 0. Significa che il nostro modello è affetto da overfitting: la rete riesce a fare predizioni accurate sulle immagini del training set ma fa fatica a generalizzare la realtà, effettuando predizioni meno precise su dati esterni al TS. Nella sezione 3.1 abbiamo citato un dataset precedente a quello attuale affetto anch'esso da overfitting; in quel caso però, il problema era presente con un'intensità maggiore. Il tempo impiegato per effettuare l'ultima sessione di training è stato sorprendentemente breve: il modello è stato trainato in meno di 1 ora.

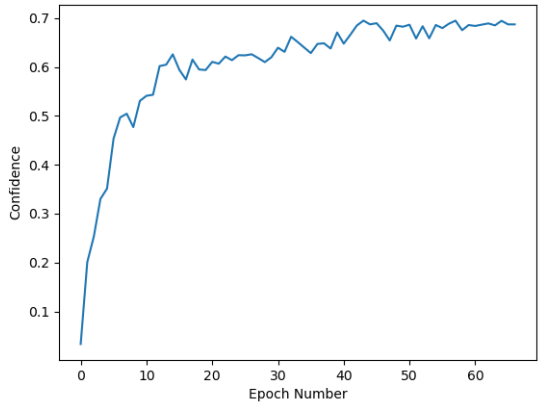


Figure 4.2: *Confidence function*

Un'altra funzione meno utile rispetto alle precedenti ma comunque interessante per la valutazione delle prestazioni è la *confidence function*. Questa funzione indica approssimativamente quanto la rete è sicura nel fare le predizioni. Quando noi essere umani diciamo "Sono sicuro...", "Non sono pienamente sicuro...", ecc. esprimiamo una sorta di confidenza. Lo stesso può fare la rete: ad ogni fine iterazione viene calcolato un valore di confidenza il cui andamento è rappresentato in fig.4.2. Un'equivalenza tra le espressioni umane e quelle una rete neurale artificiale (riguardo i livelli di confidenza) possono essere: "Sono sicuro..." $\Leftrightarrow$ 100%, "Potrei sbagliarmi ma..." $\Leftrightarrow$ 70%, "Non sono così sicuro..." $\Leftrightarrow \leq$ 50%. Il valore di confidenza a cui satura la nostra funzione è prossimo al 70%, questo significa che la rete ha ancora margini di miglioramento.

4.2 Fase di estrazione delle silhouette

Nella valutazione delle prestazioni di questa fase metteremo prima di tutto in evidenza alcuni problemi relativi all'estrazione delle silhouette, analizzando le situazioni tipiche in cui si presentano. In aggiunta, condivideremo i tempi di estrazione che abbiamo calcolato per valutare la velocità effettiva di questa parte applicativa.

I problemi sopra citati riguardano alcuni casi in cui il sistema non riesce ad estrarre una buona silhouette. Nei casi in cui si verifica il problema, è possibile identificare le cause guardando direttamente all'immagine RGB. Dalla fig.4.3 possiamo

(a) Immagine RGB di Spot

(b) Silhouette imprecisa di Spot

Figure 4.3: *Esempio di estrazione imprecisa di una silhouette*

notare come la natura circostante (vicina a Spot) disturbi il processo di estrazione. Altre situazioni che abbiamo capito essere critiche per questa fase sono caratterizzate da: particolari effetti di ombre, la presenza di più di un cane e l'eccessiva distanza del cane dalla telecamera. Tutte le imprecisioni derivanti da queste situazioni pensiamo siano causate dall'utilizzo di una versione preimpostata del software DeepLabv3+. Le migliorie future possibili verranno descritte nella sezione 5.2.2.

Per effettuare un'analisi delle prestazioni temporali abbiamo preso un set di 30 immagini e abbiamo calcolato il tempo totale che il sistema impiega per estrarre la silhouette da ognuna di esse. Per un set di 30 immagini a risoluzione 1920x1080, il sistema impiega circa 30.6 secondi. Tenendo presente che il tempo di elaborazione di un'immagine varia significativamente solo in base alla sua risoluzione (e nel set la risoluzione è sempre la stessa) possiamo affermare che in questo caso, il sistema lavora 1.02 secondi per ogni immagine. Abbiamo provato a calcolare il tempo anche su un set (sempre di 30 immagini) a risoluzione 1280x720 trovando un valore complessivo di circa 18.9 secondi; quindi per ogni singola immagine il sistema lavora per 0.63 secondi. Detto questo (sapendo che a risoluzione più basse l'estrazione è più imprecisa) bisogna tener conto del trade off tra velocità e qualità di estrazione.

4.3 Fase di predizione dei joint

Di seguito faremo qualche esempio di predizioni su dati esterni al TS mettendo inoltre in luce il problema di overfitting citato nella sezione 4.1. Infine condivideremo i tempi di predizione per valutare le prestazioni temporali.

Dagli esempi 4.4, 4.5 e 4.6 possiamo notare come si comporta la rete su immagini esterne al training set. Nell'esempio 1 la rete ha fatto un errore nel predire la posizione del joint relativo alla zampa posteriore sinistra: l'ultimo joint della zampa (che dovrebbe andare a posizionarsi sul piede), è stato posizionato invece sul piede della gamba anteriore destra. Complessivamente, il risultato di questa predizione possiamo valutarlo sufficientemente positivo. Nell'esempio 2 la rete ha predetto perfettamente la posizione di tutti i joint; questa predizione è valutabile estremamente positiva. Nell'esempio 3 abbiamo provato a vedere come si comporta la rete con

un'immagine di un cane diverso da Spot. Il risultato è una serie di imprecisioni sul posizionamento dei joint che evidenzia fortemente il problema di overfitting: la rete non è riuscita a generalizzare la realtà su cani poco più dissimili da Spot. Le

Figure 4.4: *Esempio 1 di predizione su dato esterno al TS*

Figure 4.5: *Esempio 2 di predizione su dato esterno al TS*

Figure 4.6: *Esempio 3 di predizione su dato esterno al TS [Source immagine]*

migliorie possibili relative alla precisione di predizione sono strettamente legate alla qualità del TS di cui si parla nella sezione 5.2.1.

Per la valutazione delle prestazioni temporali di questa fase abbiamo utilizzato un set di 30 silhouette: la rete ha impiegato un totale di 4.46 secondi per effettuare tutte le predizioni. Per ogni singola immagine quindi vengono spesi circa 0.15 secondi. A confronto con la fase di estrazione delle silhouette, questa è 4.2 volte più veloce.

Capitolo 5

Conclusioni

5.1 Riassunto e conoscenze acquisite

Prima di descrivere le migliorie possibili per questo progetto facciamo una breve sintesi di ciò che è stato fatto fin'ora e un resoconto delle conoscenze acquisite.

La milestone più importante da raggiungere (nonché la più difficile) è sicuramente la creazione di una rete neurale artificiale capace di effettuare previsioni con una precisione quasi del 100%. Per tentare di raggiungere tale obiettivo siamo passati per diverse fasi: lo studio delle reti neurali artificiali (per capire come manipolarle), la ricerca e l'implementazione di un modello preesistente, la creazione e ristrutturazione continua del training set e la fase di addestramento. Dopo tutti questi passaggi siamo riusciti a creare un modello che però non è quello ideale sopra citato: esso è capace di fare predizioni con sufficiente precisione su sole immagini di Spot. Possiamo però affermare che il nostro è un buon modello sperimentale di partenza. Il raggiungimento dell'obiettivo principale potrà essere possibile con le migliorie che spiegheremo successivamente.

Attraverso questo studio ritengo di aver acquisito un buon bagaglio di conoscenze nel campo dell'intelligenza artificiale; in particolare nell'ambito del deep learning. Tra le conoscenze acquisite più importanti cito sicuramente il fatto di aver compreso la struttura e il funzionamento generale delle reti neurali artificiali. Inoltre ritengo importante aver capito come manipolarle al meglio: con la creazione di un buon training set e l'impostazione di un efficace addestramento.

5.2 Direzioni future

In questo capitolo presenteremo le migliorie possibili applicabili al sistema. Alcune parti del progetto, com'è stato anche specificato in certi capitoli, andrebbero ancora migliorate al fine di eliminare i principali problemi da cui è affetto il sistema.

5.2.1 Training set più completo

Come abbiamo già detto nella sezione 3.1, il training set attuale è formato solamente da immagini di Spot. Il limite funzionale della rete portato da questa scelta è stato evidenziato nella sezione 4.3 dove si è visto che la rete non è ancora addestrata adeguatamente per fare predizioni su cani diversi da Spot.

Per effettuare un upgrade della rete e renderla utilizzabile in campo reale (nel quale sono presenti diverse tipologie di cani) andrebbe ristrutturato il TS. Per prima cosa bisognerebbe aggiungere molte più immagini annotate di quante ce ne siano adesso e soprattutto andrebbero mostrati alla rete cani diversi da Spot. Un TS di questo tipo stimiamo debba avere almeno 300 immagini (con silhouette e annotazioni) per ogni tipologia di cane. La tipologia può essere definita in base ad esempio alla grandezza, al tipo di pelo e alla struttura fisica. A questo punto, con grande probabilità si riuscirebbe ad eliminare il problema di overfitting.

Un upgrade diverso da quello appena descritto potrebbe riguardare l'utilizzo esclusivo delle immagini RGB (senza quindi preoccuparsi di estrarre le silhouette). Un lavoro interessante nel quale viene utilizzato il modello a scheletro con joint direttamente su immagini RGB è il seguente [5]: in questo lavoro viene addestrata una rete neurale artificiale a ricostruire la posa di una mano avendo come input la sola immagine RGB. Come si può intuire, questo tipo di approccio è più facilmente applicabile a realtà semplici. Se venisse richiesta la ricostruzione della posa di un cane o di un essere umano con questa metodologia: non solo si dovrebbe creare un TS non banale, ma si dovrebbero utilizzare particolari tecniche di *computer vision* oltretutto all'avanguardia.

5.2.2 Estrattore di silhouette più preciso

Nella sezione 4.2 abbiamo evidenziato alcuni casi in cui il sistema non estraeva correttamente la silhouette. Questo problema, come abbiamo già detto, pensiamo possa derivare da una versione preimpostata di DeepLabv3+ che evidentemente non è calibrata per la realtà che vogliamo studiare. Detto questo, il motivo per il quale questa versione non è pienamente efficace potrebbe essere dovuto al fatto che tale software non si dedica esclusivamente al soggetto per noi di unico interesse, cioè il cane. Una miglioria che aumenterebbe le prestazioni generali del sistema potrebbe essere quindi impostare DeepLabv3+ in modo tale da fargli studiare solo i soggetti canini.

5.2.3 Analisi dei joint

L'analisi dei joint è sicuramente un componente del sistema essenziale per il suo funzionamento. Come abbiamo già detto nella sezione 3.2.2, questa parte applicativa è ancora in fase di sviluppo.

Per distinguere se la posa di un cane sia da segnalare o meno, abbiamo pensato a due metodi diversi di risoluzione: il primo prevede di creare l'applicativo *joint analyzer* il quale sarebbe capace di analizzare la posizione dei joint attraverso calcoli puramente matematici e restituire in output la decisione. Il secondo metodo

prevede di insegnare direttamente alla rete a riconoscere ambedue i comportamenti (posizione scorretta & posizione ok); la rete stessa a questo punto sarebbe in grado di restituire in output la decisione, escludendo quindi la creazione dell'applicativo *joint analyzer*. Un giusto approccio per quest'ultimo metodo potrebbe essere: preparare un TS come spiegato nella sezione 3.1 implementando in aggiunta ad ogni immagine un tag ("Posizione da segnalare" o "Posizione ok"); successivamente trainare la rete sul nuovo TS. L'utilizzo dei joint non è escluso che possa rivelarsi superfluo (una volta adottato questo metodo) dal momento che la rete potrebbe effettuare una decisione guardando solo alla silhouette. Dei metodi sopra descritti, non abbiamo dati reali per confrontare le loro prestazioni quindi non possiamo dire con certezza quale sia il migliore.

5.3 Ringraziamenti

Grazie a questo progetto ho potuto toccare con mano la vera potenza delle reti neurali artificiali facendomi oltretutto immaginare cosa si potrebbe fare in un futuro prossimo. Il fatto che questa tecnologia (oltre ad essere in una continua fase di sviluppo) è molto interessante e soprattutto utile per migliorare il mondo in cui viviamo, mi spinge a non terminare la mia esperienza con essa una volta finito tale progetto. Oltre al progetto in se, ringrazio il mio relatore Enoch Peserico con cui ho condiviso questa esperienza; lo ringrazio inoltre per avermi dato degli spunti, consigli, consultazioni e piste da seguire per proseguire nel lavoro. Specialmente però, lo ringrazio per gli insegnamenti preziosi che mi ha dato. Non posso inoltre non ringraziare il mio cane Spot, senza il quale avrei avuto difficoltà nel trovare materiale per lo studio. Infine ringrazio la mia famiglia e il supporto che mi ha dato, senza il quale non sarei qui a scrivere questa tesi.

Bibliografia

[1] *Anaconda Software Distribution*. Version Vers. 2-2.4.0. 2020. URL: https://docs.anaconda.com/.

[2] Benjamin Biggs et al. *Creatures great and SMAL: Recovering the shape and motion of animals from video*. 2018. arXiv: 1811.05804 [cs.CV].

[3] Liang-Chieh Chen et al. "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation". In: *CoRR* abs/1802.02611 (2018). arXiv: 1802.02611. URL: http://arxiv.org/abs/1802.02611.

[4] Statista Research Department. "Number of dogs in Italy from 2014 to 2020. (English)". In: (2021). URL: https://www.statista.com/statistics/515521/dog-population-europe-italy/.

[5] Liuhao Ge et al. *3D Hand Shape and Pose Estimation from a Single RGB Image*. 2019. arXiv: 1903.00812 [cs.CV].

[6] Jacob M Graving et al. "DeepPoseKit, a software toolkit for fast and robust animal pose estimation using deep learning". In: *eLife* 8 (2019), e47994. URL: https://doi.org/10.7554/eLife.47994.

[7] W.S. McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity. (English)". In: *Bulletin of Mathematical Biology* 5 (2016), pp. 115–133. DOI: https://doi.org/10.1007/BF02478259.

[8] Alejandro Newell, Kaiyu Yang, and Jia Deng. *Stacked Hourglass Networks for Human Pose Estimation*. 2016. arXiv: 1603.06937 [cs.CV].

[9] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain. (English)". In: *Psychological Review* 65.6 (1958). URL: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.335.3398&rep=rep1&type=pdf.